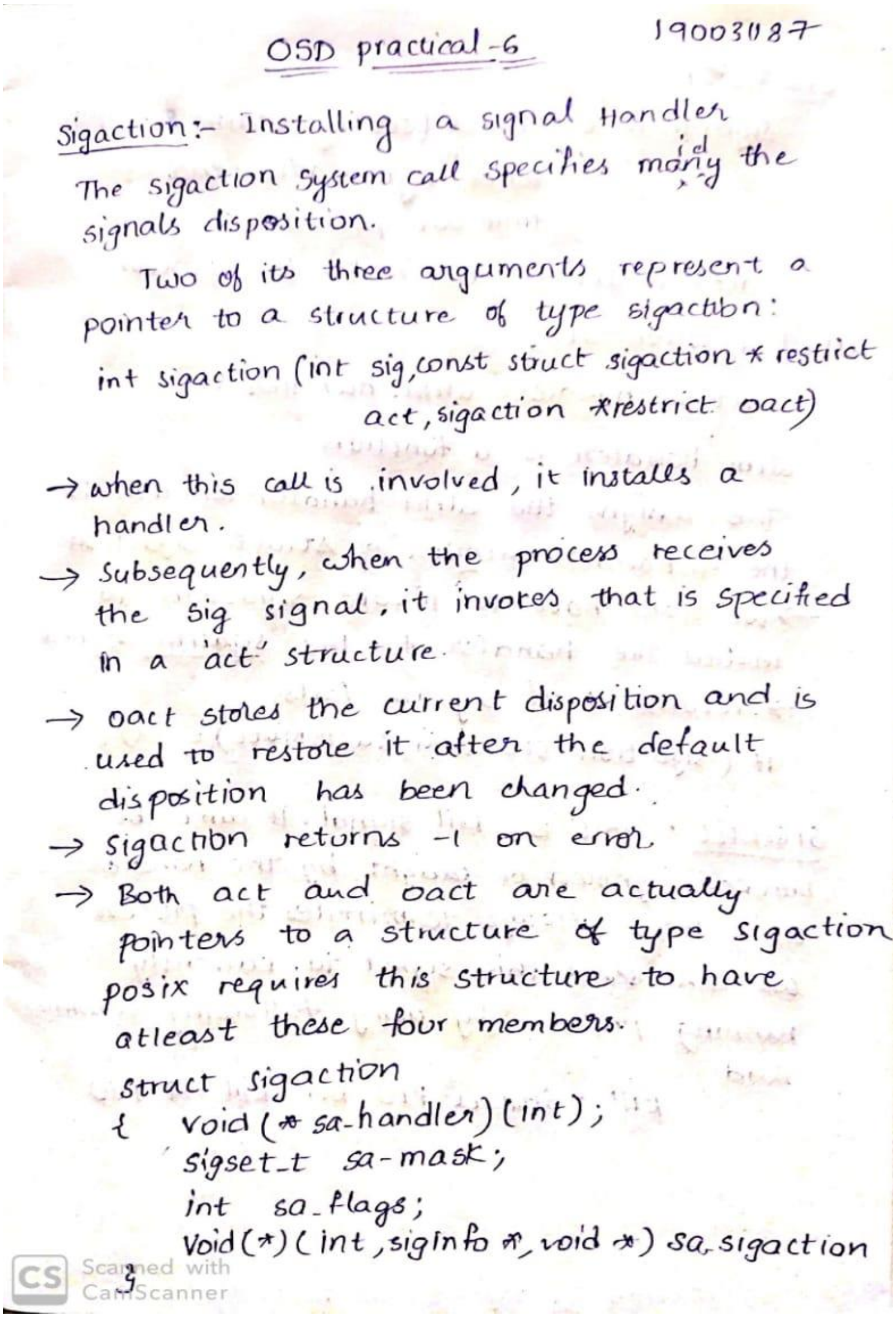
Operating System and Design (19CS2106S)
Lab- 6
Pre-Lab

POSIX signals. sigaction: This call specifies the signal handler. Two of the arguments to this call specify a structure that is also named sigaction. alarm: The alarm call is used in the next example to set a timer that generates the SIGALRM signal after the timeout period. The library function sleep uses alarm.pause: This is somewhat like the shell’s read statement. It holds up program execution until a signal is received. kill: You can send a signal to a process using this system call. A library function, raise, uses kill to send any signal to the current process.



OSD practical-6 190031187

Sigaction:- Installing a signal Handler

The sigaction system call specifies modify the signals disposition.

Two of its three arguments represent a pointer to a structure of type sigaction:

```
int sigaction (int sig,const struct sigaction * restict
                act, sigaction *restrict oact)
```

→ when this call is involved, it installes a handler.

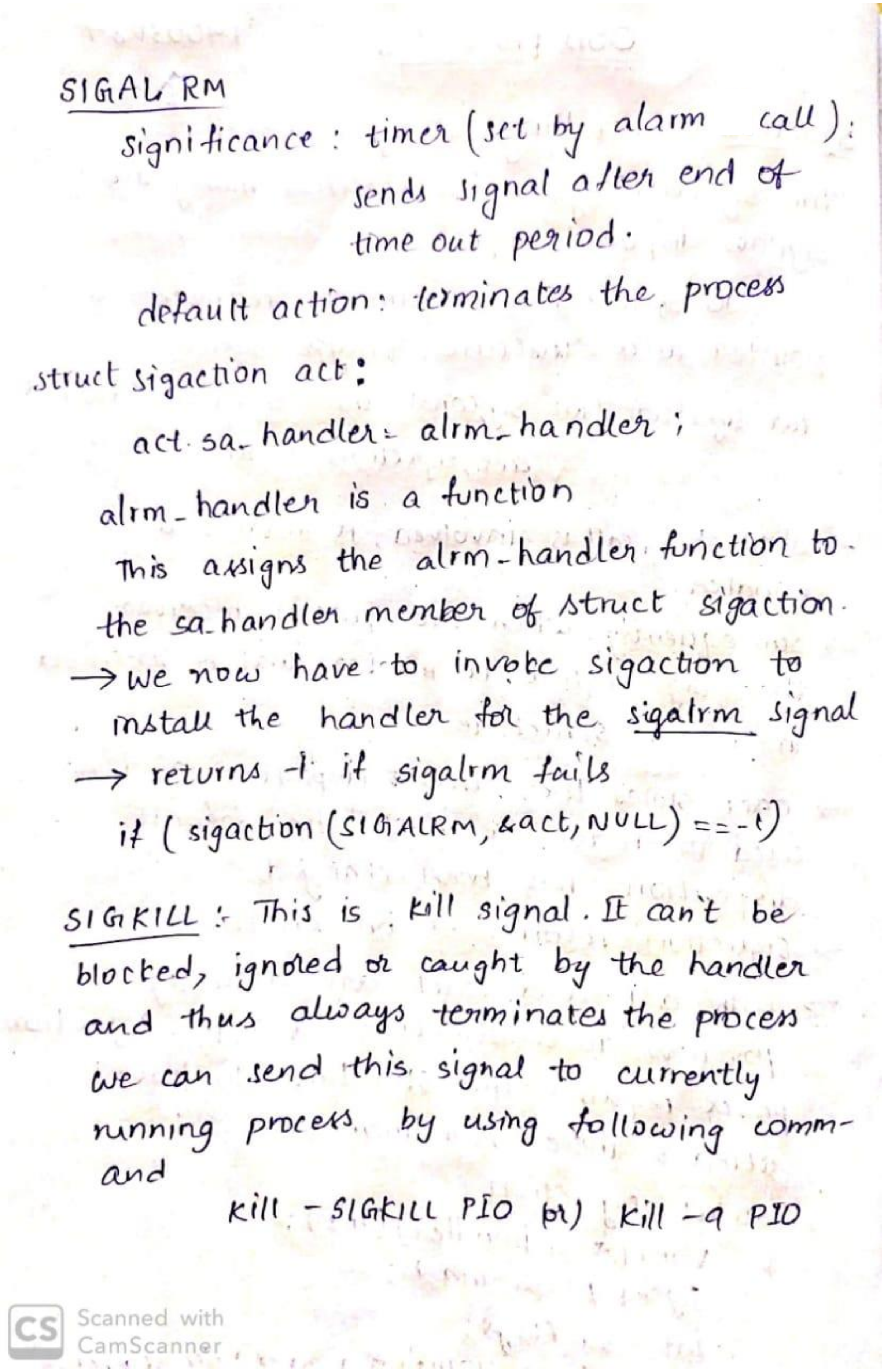
→ Subsequently, when the process receives the sig signal, it invokes that is specified in a 'act' structure.

→ oact stores the current disposition and is used to restore it after the default disposition has been changed.

→ sigaction returns -1 on error

→ Both act and oact are actually pointers to a structure of type sigaction posix requires this structure to have atleast these four members.

```
struct sigaction
{   void (* sa-handler)(int);
    sigset_t sa-mask;
    int sa_flags;
    void(*)(int,siginfo *, void *) sa_sigaction
}
```



SIGAL RM

significance : timer (set by alarm call): sends signal after end of time out period.

default action: terminates the process

struct sigaction act;

act.sa_handler= alrm_handler;

alrm_handler is a function

This assigns the alrm-handler function to the sa_handler member of struct sigaction.

→ we now have to invoke sigaction to install the handler for the sigalrm signal

→ returns -1 if sigalrm fails

if (sigaction (SIGALRM, &act, NULL) ==-1)

SIGKILL :- This is kill signal. It can't be blocked, ignored or caught by the handler and thus always terminates the process we can send this signal to currently running process by using following command

kill -SIGKILL PID (or) kill -9 PID

In-Lab

1. signal.c -- Waits for 5 seconds for user input and then
2. Generates SIGALRM that has a handler specifiedkillproce ss.c -- Uses fork and exec to run a user-defined programand kills it if it doesn’t complete in 5 seconds.

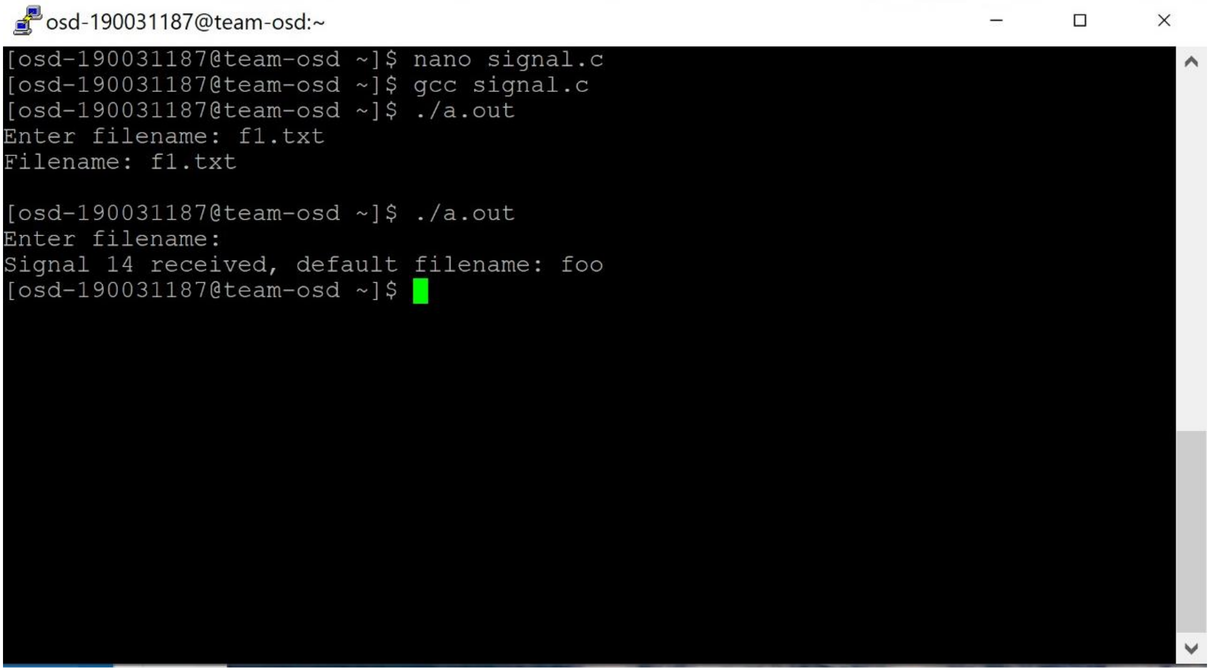
1) signal.c :

Code:

```
#include <stdio.h>
#include <sys/stat.h> /* For struct stat */
#include <stdarg.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <signal.h> #define BUFSIZE 100 void
alrm_handler(int signo); /* Prototype declaration */
char buf[BUFSIZE] = "foo\0"; /* Global variable */
void quit(char *message, int exit_status)
{
printf(" %s",message);
exit(exit_status);
}

int main (void) { int n; struct sigaction act; act.sa_handler
= alrm_handler; /* Specify handler */ if
(sigaction(SIGALRM, &act, NULL) == -1) /* Install handler
*/ quit("sigalrm", 1); fprintf(stderr, "Enter filename: ");
alarm(5); /* Set alarm clock; will deliver */
n = read(STDIN_FILENO, buf, BUFSIZE); /* SIGALRM in 5 seconds
*/ if (n > 1) /* Will come here if user inputs */
fprintf(stderr, "Filename: %s\n", buf); /* string within 5 seconds */
exit(0);
}
void alrm_handler(int signo)
{
fprintf(stderr, "\nSignal %d received, default filename: %s\n", signo,
buf); exit(1); }
```

Output:-



```
[osd-190031187@team-osd ~]$ nano signal.c
[osd-190031187@team-osd ~]$ gcc signal.c
[osd-190031187@team-osd ~]$ ./a.out
Enter filename: f1.txt
Filename: f1.txt

[osd-190031187@team-osd ~]$ ./a.out
Enter filename:
Signal 14 received, default filename: foo
[osd-190031187@team-osd ~]$
```

2) killprocess.c:

Code:-

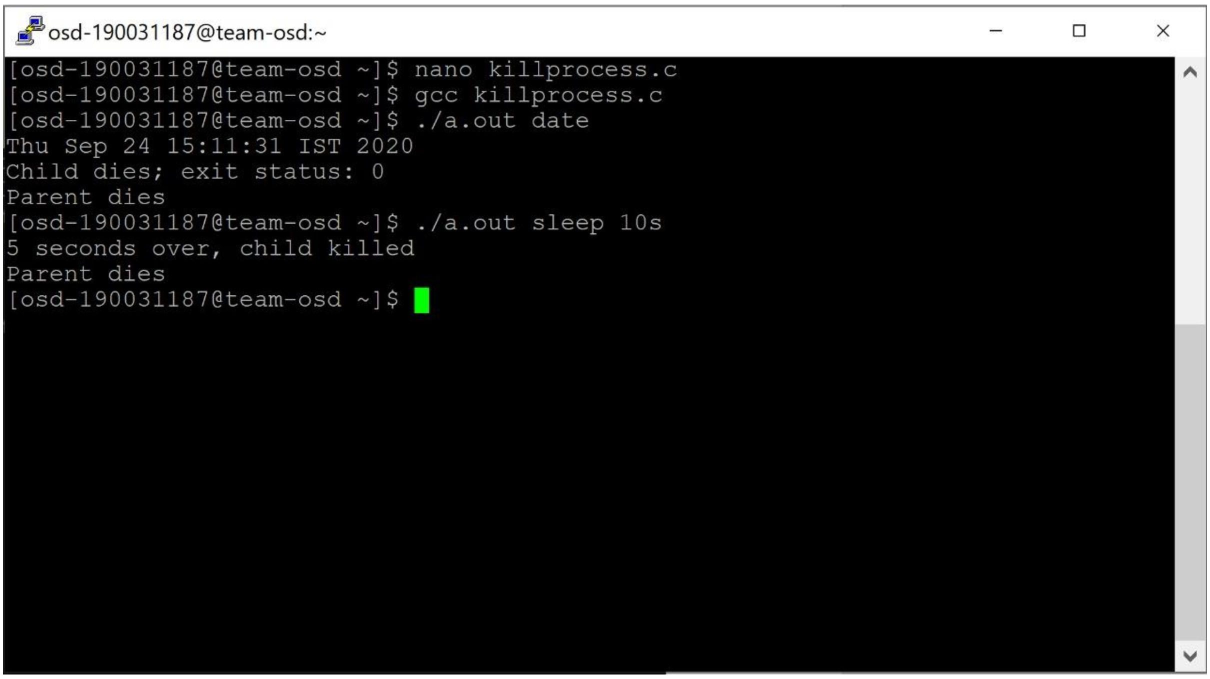
```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdlib.h>

pid_t pid;
int main (int argc, char **argv) {
int i, status;
void death_handler(int signo); /* A common signal handler this time */

struct sigaction act; act.sa _handler = death_handler;
sigaction(SIGCHLD, &act, NULL); /* Disposition for these two
signals */ sigaction(SIGALRM, &act, NULL); /* set to enter a single
handler */

switch (pid = fork()) { case -1: fprintf(stderr, "Fork
error\n"); case 0: execvp(argv[1], &argv[1]); /* Execute
command */ perror("exec"); break; default: alarm(5);
/* Will send SIGALRM after 5 seconds */ pause(); /*
Will return when SIGCHLD signal is received */
fprintf(stderr, "Parent dies\n");
}
exit(
0);
}
void death_handler(int signo) { /* This common handler picks up the */
int status; /* exit status for normal termination */ /* but sends
the SIGTERM signal if */ switch (signo) { /* command doesn’t
complete in 5 seconds */ case SIGCHLD: waitpid(-1, &status, 0);
/* Same as wait(&status); */
fprintf(stderr, "Child dies; exit status: %d\n",
WEXITSTATUS(status)); break; case
SIGALRM: if (kill(pid, SIGTERM) == 0)
fprintf(stderr, "5 seconds over, child killed\n");
}
}
```

Output:-



```
[osd-190031187@team-osd ~]$ nano killprocess.c
[osd-190031187@team-osd ~]$ gcc killprocess.c
[osd-190031187@team-osd ~]$ ./a.out date
Thu Sep 24 15:11:31 IST 2020
Child dies; exit status: 0
Parent dies
[osd-190031187@team-osd ~]$ ./a.out sleep 10s
5 seconds over, child killed
Parent dies
[osd-190031187@team-osd ~]$
```

Post-Lab

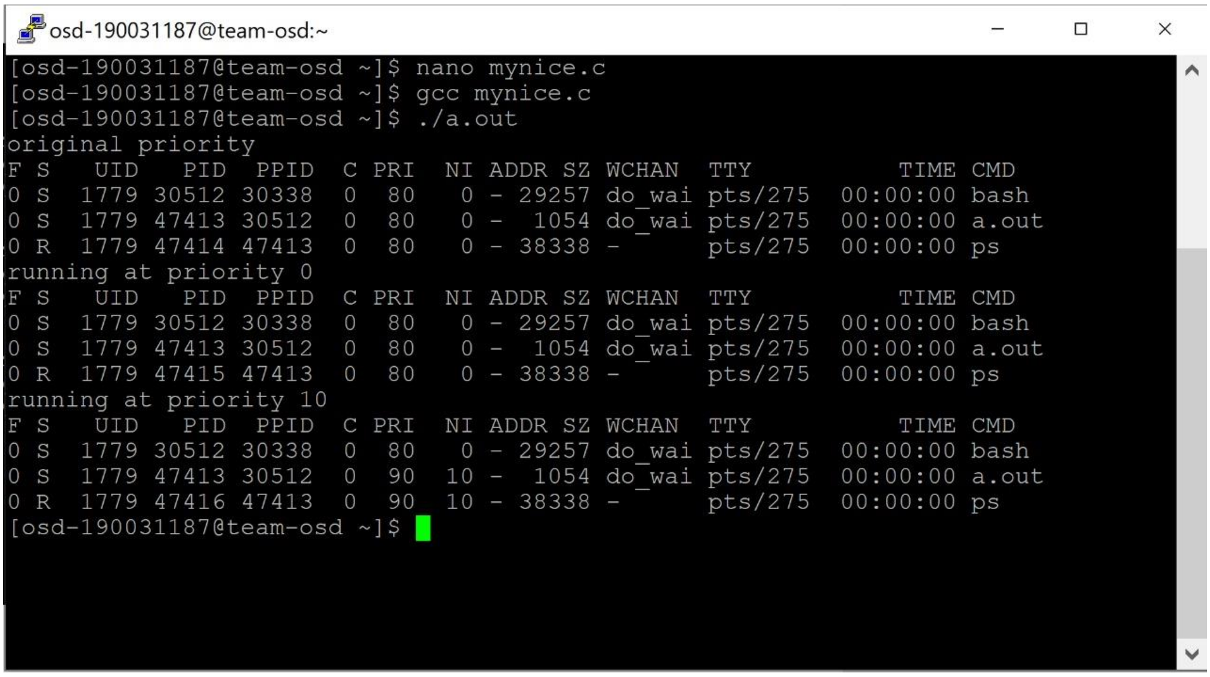
1. mynice.c: A child process inherits its priority value from its parent, and change it by using nice ()
2. program to demonstrate time and times System Call.

mynice.c:

Code:

```
#include <stdio.h>
main ()
{
printf ("original priority\n"); system
("ps -l"); /* Execute a ps */ nice (0); /*
Add 0 to my priority */ printf
("running at priority 0\n"); system
("ps -l"); /* Execute another ps */ nice
(10); /* Add 10 to my priority */ printf
("running at priority 10\n");
system ("ps -l"); /* Execute the last ps */
}
```

Output:



```
[osd-190031187@team-osd ~]$ nano mynice.c
[osd-190031187@team-osd ~]$ gcc mynice.c
[osd-190031187@team-osd ~]$ ./a.out
original priority
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1779 30512 30338  0  80   0 - 29257 do_wai pts/275  00:00:00 bash
0 S  1779 47413 30512  0  80   0 -  1054 do_wai pts/275  00:00:00 a.out
0 R  1779 47414 47413  0  80   0 - 38338 -      pts/275  00:00:00 ps
running at priority 0
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1779 30512 30338  0  80   0 - 29257 do_wai pts/275  00:00:00 bash
0 S  1779 47413 30512  0  80   0 -  1054 do_wai pts/275  00:00:00 a.out
0 R  1779 47415 47413  0  80   0 - 38338 -      pts/275  00:00:00 ps
running at priority 10
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1779 30512 30338  0  80   0 - 29257 do_wai pts/275  00:00:00 bash
0 S  1779 47413 30512  0  90  10 -  1054 do_wai pts/275  00:00:00 a.out
0 R  1779 47416 47413  0  90  10 - 38338 -      pts/275  00:00:00 ps
[osd-190031187@team-osd ~]$
```

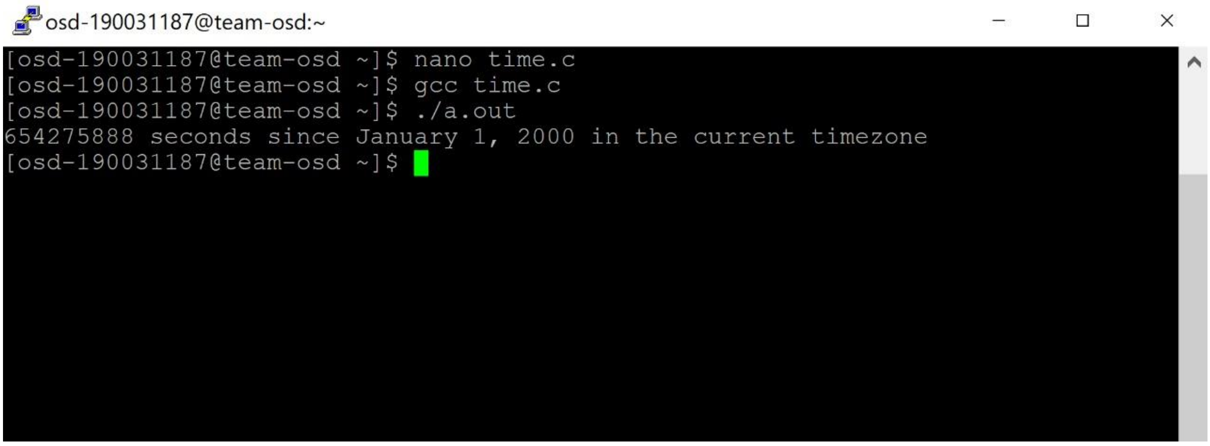
2) **time.c:**

Code:

```
#include <stdio.h> /* printf */
#include <time.h> /* time_t, struct tm, difftime, time, mktime */

int main ()
{
time_t timer; struct tm y2k = {0}; double seconds;
y2k.tm_hour = 0; y2k.tm_min = 0; y2k.tm_sec = 0;
y2k.tm_year = 100; y2k.tm_mon = 0; y2k.tm_mday = 1;
time(&timer); /* get current time; same as: timer =
time(NULL) */ seconds = difftime(timer,mktime(&y2k));
printf ("%.f seconds since January 1, 2000 in the current timezone", seconds);
return 0;
}
```

Output:



```
[osd-190031187@team-osd ~]$ nano time.c
[osd-190031187@team-osd ~]$ gcc time.c
[osd-190031187@team-osd ~]$ ./a.out
654275888 seconds since January 1, 2000 in the current timezone
[osd-190031187@team-osd ~]$
```

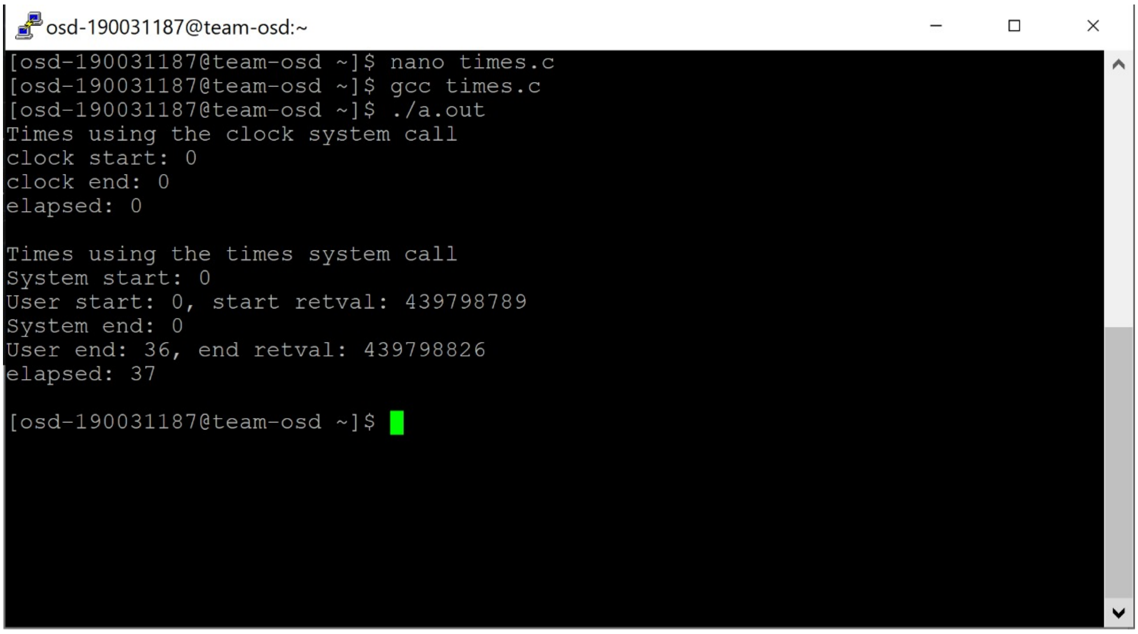
times.c:

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/times.h>
#include <time.h>

int main() { struct tms times_start,
times_end; clock_t times_start_retval,
times_end_retval; clock_t clock_start,
clock_end;
int i;
/* clock called first and last, so estimates using "clock" should
be slightly longer than estimates using "times" */
if((clock_start = clock()) == -1) { perror("starting clock");
return -1;
}
if((times_start_retval = times(&times_start)) == -1) {
perror("starting times");
return -1;
}
for(i = 100000000; i; i--); // do work
if((times_end_retval = times(&times_end)) == -
1) { perror("ending timer"); return -1;
}
printf("Times using the clock system call\n"); printf("clock
start: %li\nclock end: %li\n", clock_start, clock_end);
printf("elapsed: %li\n\n", clock_end - clock_start);
printf("Times using the times system call\n"); printf("System
start: %li\nUser start: %li, start retval: %li\n",
times_start.tms_stime, times_start.tms_utime,
times_start_retval); printf("System end: %li\nUser end: %li, end
retval: %li\n", times_end.tms_stime, times_end.tms_utime,
times_end_retval); printf("elapsed: %li\n\n", times_end_retval -
times_start_retval);
return 0;
}
```

Output:



```
[osd-190031187@team-osd ~]$ nano times.c
[osd-190031187@team-osd ~]$ gcc times.c
[osd-190031187@team-osd ~]$ ./a.out
Times using the clock system call
clock start: 0
clock end: 0
elapsed: 0

Times using the times system call
System start: 0
User start: 0, start retval: 439798789
System end: 0
User end: 36, end retval: 439798826
elapsed: 37

[osd-190031187@team-osd ~]$
```